Assisting Data Warehousing Populating Processes Design through Modelling using Coloured Petri Nets

Diogo Silva, João M. Fernandes and Orlando Belo
Algoritmi R&D Centre, University of Minho, Braga, Portugal
diogosantossilva@gmail.com, jmf@di.uminho.pt, obelo@di.uminho.pt

Keywords: Data Warehousing Systems, Populating Processes, Modelling and Simulation, Coloured Petri Nets, Change Data Capture Process.

Abstract: Data warehousing systems populating processes are responsible for loading their data repositories – the data warehouses – with information they extract from operational sources. The tasks that integrate these processes are the most complex ones that we can find in a data warehousing system. For a flawless implementation, modelling these processes previously is important so that a correct set of requirements is considered. This paper approaches conceptual modelling and simulation of the populating processes of a DWS, by applying Coloured Petri Nets in the design of independent populating tasks. We adopt a change data capture task as the case study in order to demonstrate the effective application of coloured petri nets for modelling and simulating data warehousing populating processes.

1 INTRODUCTION

Any *Data Warehousing System* (DWS) integrates a highly specialized "single" repository – a *data warehouse* (DW) - containing high quality data that are detailed, historic, subject oriented and non-volatile (Inmon, 1996). ETL (Extract-Transform-Load) processes are responsible for populating DWs, extracting data from distinct operational sources, in different formats, applying a series of cleansing and transformation operations in order to make these data consistent, correctly structured and error-free. These types of processes usually take place in an intermediate storage area, a *Data Staging Area* (DSA), designed specifically to sustain this kind of operations, After the successful execution of all the defined ETL processes, if suitable, the data previously extracted is finally loaded into the DW. The extracted data can contain errors such as duplicate data, invalid or wrong values, or other inconsistent values, often caused by simple typing errors or transformation functions badly designed and programmed. It is fundamental to assure that all the data flaws are corrected prior to their insertion in the DW. A group of operations (or transformations) are then applied to the data residing in the DSA with the objective of converting the data format to the formats required by the DW, enhancing as well its overall quality. Operations, such as the conversion of measures or monetary units, calculations of derived attributes, creation and assignment of surrogate keys, matching of data from different sources, among others, are included in this ETL stage. Posteriorly, the data can then be used to refresh the DW, by rewriting part of the stored information, especially the one related to dimension tables, or by updating it with new data. A poor implementation of an ETL system may result in low quality information, which can entirely compromise a DWS (English, 1999). Furthermore, the costs associated with fixing and correcting low quality data after the implementation and deployment of the DWS are usually very high, as are the time and the involved resources. Kimball and Caserta (2004) state that an ETL system can consume up to 70% of the resources needed for its implementation and maintenance. This happens because it is one of the most complex and technically challenging processes among all of the DWS implementation phases (Golfarelli and Rizzi, 2009). Taking into account all the different operations that are implemented in ETL systems, the high learning curve presented by the available ETL implementation tools and the lack of a conceptual model operation provided by these tools, it is no surprise that several organizations choose to implement their ETL systems in an ad hoc fashion (Vassiliadis et al., 2002), and that an

important fraction of these projects fail because the final set of data does not meet the requirements of the DWS project. To overcome this situation, we believe that is necessary to adopt strong conceptual modelling and validation methodologies in the development of ETL systems, in order to design and test different operations in each stage. Work concerning conceptual modelling approaches for ETL systems is extensive, e.g. (Vassiliadis et al., 2002) (Abelló et al., 2006) (Golfarelli, 2008), but no standard language is used for this matter, and the design and the validation tasks are not supported by a proper tool.

In this paper we tackle these issues by proposing the *Coloured Petri Nets* (CPN) modelling language (Jensen and Kristensen, 2009) to be used within the design and specification of ETL processes. CPN models are very adequate to describe and study all systems that are concurrent, asynchronous, distributed, non-deterministic and stochastic. The implementation of ETL systems includes many of these characteristics, which makes CPN quite suitable to specify and analyse them, at least in a preliminary development phase. We intend to provide a well-sustained approach to specify the behaviour of ETL systems using CPN models. We present a case study in modelling and validation of ETL systems, related with the Change Data Capture (CDC) in a conventional operational system. The paper was organized as follows: section 2 presents a generic introduction about ETL modelling and CPNs; section 3, describes the behaviour of a CDC operation; and section 4 gives a closer look to the transaction log used in the CDC process selected: section 5 presents the CPN model (and its corresponding modules) that were implemented; and, finally, section 6, includes the conclusions of this paper.

2 MODELING ETL PROCESSES

Usually, implementation of a DWS is a difficult endeavour to accomplish, since errors and ambiguous requirements are recurrent. Many of these undesirable circumstances are related with the specific implementation of the ETL component of the DWS. As we said, it is one of the most important pieces of the entire system, being responsible to take data from different information sources to DWs. The ETL implementation is accomplished through a very diversified set of treatments ensuring that data arrives to the DW in perfect conditions and in accordance with the requirements of the decision-makers. Thus, it is not a surprise that such component had taken so many attentions by researchers in the field. In part, such is due to the lack of standards in the design and development of ETL processes and to the potential impact of a DWS implementation failure. According with Kimball and Caserta (2004), the development of a conventional ETL system involves two distinct paths, covered in parallel: plan and design the entire system, and deal with all the aspects of data flowing. Both paths could be modelled and simulated using CPNs, with clarity and effectively.

CPN models (Jensen, 1998) have been successfully applied in numerous industrial and academic projects worldwide in distinctive areas. They are an extension to the original Petri Nets (Petri, 1966), which are adequate to build models, for laPetrirge and complex systems, that are composed of several smaller parts, and where concurrency, communication and synchronization are important characteristics. They come to cover the inexistence of data and hierarchical concepts (Jensen and Kristensen, 2009) often used in practical real-world applications. The main advantage in building hierarchical CPN models is that each of the modules can be designed, tested and validated independently, allowing the system modeller to work both bottom-up or top-down, and also allowing for a system to be built and visualised with several levels of abstraction, resulting in smaller and more compact models. The CPN modelling language, together with CPN Tools, may bring many advantages when it comes to model and validate ETL systems. It allows to analyse and to study their behaviour through simulation, which can be used to verify their performance-wise behaviour. This approach has the potential to detect anomalies and incorrect behaviours in an early stage of the design. CPN models have been used in communication protocols and networks to model different versions of the TCP protocol (Figueiredo and Kristensen, 1999), for instance. Hewlett-Packard has also used the CPN in industrial projects to model and analyse their on-line transaction processing system (Cherkasova et al., 1993). These are just a few examples of what we can do with CPN in real-world applications. As far as we know, the CPN modelling language has not yet been applied in the design and validation of ETL systems. Different approaches to the design of ETL systems using other languages were proposed. Those approaches focus on several aspects of ETL processes, ranging from modelling the structure and content of the data stores of a DW (Vassiliadis et al., 2002) to modelling ETL activities

(Muñoz et al., 2009). Thus, taking advantage of the features provided by the CPN modelling language, we use CDC, one of the most relevant (and frequent) ETL tasks, to demonstrate the application of CPNs for modelling and validation of real-world ETL processes.

3 CDC TASKS

Commonly, CDC mechanisms are used to identify and capture data changes in the tables of the source systems, so they can be delivered to the DW. There are a number of different ways to set up this kind of mechanisms. The simpler ones consist on having dedicated columns in the tables of the operational systems, in order to track row modifications, such as timestamp, version number or status indicator columns, whose values need to be updated when a modification takes place. A CDC mechanism, although simple, must be considered in the design stage of the database of the source system, so that the tables that need to be audited possess the correct structure right from scratch. A more reliable method to implement a CDC mechanism is through the use of triggers on tables that need to be audited. The trigger acts as an automatic mechanism to record information about changes that occur in a specific transactional table into a corresponding audit table. The latter acts as an ordered queue that can subsequently be used in the target *Slowly Changing Dimension* (SCD), a special table in a DW that supports the analysis of the data of the DW with respect to a specific decision-maker perspective. This method overcomes the downsides of the previously presented method, by using audit tables on the source tables that need to be audited the impact is minimum and the dimension table's structure remains unaltered. However, this method stills is an intrusive CDC method, since the log trigger may not be implemented in the transactional source tables that need to be audited and, in many occasions, the DWS administrator may not have permissions to do so during the implementation of the current ETL process, once this is of the competence of the operational system administrator. A different manner to capture changes made in the databases of the operational sources consists in reading transaction log files to detect modifications occurred in source tables that need to be audited. This CDC mechanism is the less intrusive one, with minimal impact on source systems, since we don't need to alter any part of the operational database schema and no triggers need to be implemented in each of the tables, avoiding their negative impact on the performance of the systems. Instead, all the information about data modification is read and interpreted from the transaction log into audit tables that can be similar to the ones used in the log trigger CDC mechanism. Because of the non-intrusive manner in which the source modifications are captured, this is the CDC method adopted for our ETL process. The reading and interpretation of the transaction log presents itself as the most challenging part of the process since each *Database Management System* (DBMS) adopts a specific structure and content for its transaction log files, without providing the much needed documentation. The SQL Server 2008 was the chosen DBMS for the analysis of the structure of the transaction log file and interpretation of its content, so that, from this initial study, the corresponding CPN model could be implemented.

4 THE TRANSACTION LOG

The fn_dblog(NULL,NULL) function in Microsoft SQL Server 2008 allows us to view a transaction log as a relational table. This function has two optional parameters - the starting and ending log sequence number (LSN) -, which can be viewed as the unique identifier of each log record. By replacing the 'NULL' values with an actual LSN, we limit the number of presented log records. The first step of the CDC modelling process is to analyse the structure and content of the transaction log', as well as the implication of each of its records (transactions) created by Data Manipulation Language (DML) queries, as these are the ones responsible for update, insert and delete operations. Our transaction log has 119 columns, the majority of them possess solely 'NULL' values, and their importance is unknown because we didn't have access to the respective documentation. For this reason, we have created a view of the transaction log with the necessary and most relevant attributes to this work, for an easier analysis, and a subset of the view (Table 1). The created view has only 7 attributes, which are the ones that are needed to support modelling the selected CDC process. The LSN is the unique identifier of each transaction log record, while the T.ID (i.e., the Transaction ID) is the reference to the database transaction that has generated the log record – note that the same transaction usually creates several entries in the transaction log table. Both attributes contain hexadecimal values that can be long and difficult to

differentiate. In this example they were replaced by integer values in order to simplify the analysis and the interpretation of each record.

Table 1: A view of the transaction log.

LSN	T. ID	Operation	T. Name	End Time	AllocUnitName	RowLog Contents
1	1	LOP_BEGIN_XACT	INSERT	NULL	NULL	NULL
2	1	LOP_INSERT_ROWS	NULL	NULL	dbo.TestTable	0x10006C0
3	1	LOP_COMMIT_XACT	NULL	2012/08/03 22:57:05	NULL	NULL
4	2	LOP_BEGIN_XACT	UPDATE	NULL	NULL	NULL
5	2	LOP_MODIFY_ROW	NULL	NULL	dbo.TestTable	0x63
6	2	LOP_MODIFY_ROW	NULL	NULL	dbo.TestTable	0x63
7	2	LOP_COMMIT_XACT	NULL	2012/08/03 22:58:29	NULL	NULL
8	3	LOP_BEGIN_XACT	DELETE	NULL	NULL	NULL
(…)	(…)	(…)	(…)	(…)	(…)	(…)

The Operation attribute receives the type of operation that was performed and recorded in the log. Each transaction begins with the LOP_BEGIN_XACT operation, and ends with the LOP_COMMIT_XACT operation. The T.Name indicates the name of the transaction, which can be identified in its first log record, i.e., the LOP_BEGIN_XACT record. In the remaining records, this field is left as 'NULL'. Each transaction has an associated timestamp and the End Time is recorded in each LOP_COMMIT_XACT statement and left as 'NULL' in the remaining log records. The name of the schema and table where modifications occurred are stored in the *AllocUnitName* for log records that represent actual modifications and are left as 'NULL' in the remaining records. The types of operations, their timestamps and the name of the tables where modifications took place have been identified. So the final piece to the puzzle is the identification of the transaction log's attributes that store information about the actual modifications, that is, the attributes names, their types and values that were modified. Unfortunately, this information is not directly displayed in the transaction log. Instead, it is stored as a hexadecimal value in a series of attributes named *Row Log Contents*. There is a series of 5 Row Log Contents attributes in the transaction log, numbered from 0 to 4. They store the modified information according to the type of operation. In this case study, once again to make it simpler and more readable, merely one attribute is used to represent this series of attributes. The actual data extraction process from these attributes is out of the scope of this study but was simulated in the implemented model of the CDC process. Table 1 presents some examples of transactions and log records. The type of the first transaction can be identified in the T field of the first record. Name attribute (i.e., an insert), the following record is the one that holds the information about the inserted row, while the timestamp of the operation is presented in the End Time attribute of the first transaction's final log record. The second transaction, an update, begins in the table's fourth record; in this transaction there are two records with the LOP_MODIFY_ROW operation, meaning that two rows were updated with a single DML query. The third transaction, representing a delete operation on two rows, can be interpreted in an analogous way.

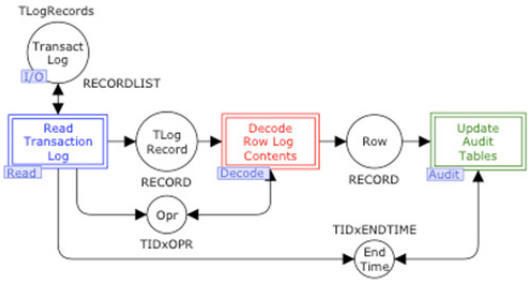

Figure 1: The CDC process prime module.

5 CAPTURING CHANGES

The CDC process is implemented using a hierarchical CPN composed of three main modules. The prime module, composed by three sub-modules, allows for a more abstract and cleaner view of the entire net (Figure 1). The first step of the CDC process is to read the transaction log of the operational system, processing records according to its operation type. That is accomplished in the Read module, represented by the Read Transaction Log substitution transition. The Decode module is responsible for extracting modified data from the *Row Log Content* attribute of the record, which is represented by the *Decode Row Log Contents* substitution transition. Finally, in the Audit module, represented by the *Update Audit Tables* substitution transition, the audited data is inserted in the corresponding audit table. In addition to these substitution transitions, the prime module is also formed by four places: 1) *Opr* that is the output socket of the *Read Transaction Log* substitution

transition and the input/output socket of the *Decode Row Log Contents* substitution transition; 2) *End Time* that is the second output socket of *Read Transaction Log*, which is also the input/output socket of *Update Audit Tables*; 3) *TLog Record*, that is used to model a transaction log record; is the third output socket of *Read Transaction Log* and the input socket of *Decode Row Log Contents*; and 4) *Row*, that models a log record after the decoding of the *Row Log Contents*, acting as the output socket of *Decode Row Log Contents* and input socket of *Update Audit Tables*. The TIDxOPR and TIDxENDTIME colour sets represent the product of an integer and a string, being implemented as:

```
colset NO = int;
colset ST = string;
colset TIDxENDTIME = product NO * ST;
colset TIDxOPR = product NO * ST;
```

The RECORD colour set is used to model a record from a relational database and it is defined as the union of the different types of records used in this CDC process:

```
colset RECORD = union TLogRec:TLOGREC +
AudRec:AUDREC +  DecRec:DECREC;
```

In order to make the model more uniform, all the places that represent a relational table (e.g., an audit table) or individual records have this colour set. The colour sets that are part of this union and represent the existing types of records are described in the following sections.

5.1 The Read Module

The Read module (Figure 2) is responsible for the extraction and initial processing of the records of the transaction log according to their type of operation. In our case, three or more records with different operations compose each transaction. The log record with the LOP_BEGIN_XACT operation marks the beginning of the transaction and holds the name of the transaction (insert, update or delete), the record with the LOP_COMMIT_XACT operation is the final record of every transaction and also holds information on its end time. The LOP_INSERT _ROWS, LOP_UPDATE_ROWS and the LOP_DELETE_ROWS operations belong to the log records holding the information about the actual row modification in the operational sources. A single transaction is responsible solely for one kind of operation (insert, delete or update), but the same operation can occur in several different records.

This module is composed by five places and three transitions. Three of the places – *Opr*, *TLog Rec* and *End Time* – are the output ports of the

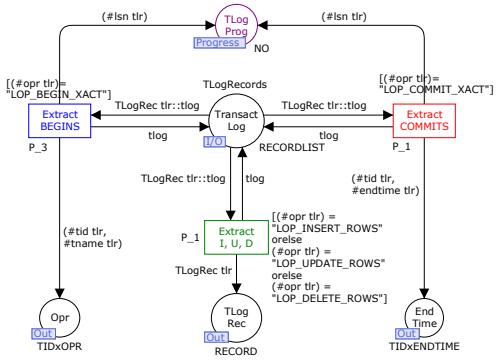

Figure 2: The Read module.

module. They were already described previously. The fusion place *TLog Prog* of the colour set NO is used to manage the progress of this CDC process by saving the LSN of the last log record to be fully processed, which is necessary to resume the process should it terminate unexpectedly. The last place, Transact Log of the colour set RECORDLIST, is used to model the transaction log view presented previously. In spite of representing a relational table, the colour set RECORDLIST is used exceptionally, instead of RECORD, so that the log records can be processed sequentially as a FIFO list. This colour set is implemented as:

```
colset RECORDLIST = list RECORD;
```

The three existing transitions are used to extract individual log records from the place Transact Log according to the records' type of operation. The arc expression TLogRec tlr::tlog is used to extract the head of the list (i.e., a log record) into the corresponding transition, while the remaining list is returned to Transact Log through the arc expression tlog1:

```
var tlog: RECORDLIST;
var tlr : TLOGREC;
```

The colour set TLOGREC is used to model the transaction log record, a record with seven fields that is defined as:

```
colset TLOGREC = record lsn:NO * tid:NO *
opr:ST * tname:ST * endtime:ST *
aun:ST * rlc:ST;
```

The *lsn* and *tid* fields, with colour set NO, are integers used to represent the LSN and the Transaction ID, respectively. The remaining fields – *tname*, *opr*, *endtime*, *aun* and *rlc* – with colour set ST, are strings that represent the Transaction Name (i.e., insert, update or delete), the operation of the log record, the *End Time* of the transaction, the *Alloc Unit Name* and the Row Log Contents of the log record. The Extract BEGINS transition is

responsible for extracting the first record of each transaction, i.e., the records with the LOP_BEGIN_XACT operation, through the guard expression `[(#opr tlr) = "LOP_BEGIN_XACT"]`. From this point, the ID and the name of the transaction are passed to the output port *Opr*, through the arc expression (#tid tlr, #tname tlr), and the place *TLog Prog* is updated with the LSN of the processed record through the arc expression (#lsn tlr). The importance of this first record is to mark the beginning of a new transaction, as well as to preserve the operation of the transaction in the place *Opr*, so that the remaining log records (i.e., the ones that contain the Row Log Contents values) can be processed accordingly in the Decode module. The "Extract I, U, D" transition is responsible for the extraction of all the log records that are neither BEGINs nor COMMITs. The guard expression allows the transition to activate if the operation of the log record is LOP_INSERT_ROWS, LOP_MODIFY _ROWS or LOP_DELETE_ROWS. The records extracted through this transition are passed to the output port *TLog Rec* so that the information contained in the *rlc* field can be decoded in the next module. The final piece of information needed to construct an audit record is the end time of the transaction. This information is presented in final record of every transaction that is extracted from Transact Log through the Extract COMMITS transition. From this transition, the place End Time is updated with the timestamp of the transaction's commit operation as well as the transaction ID, so that this timestamp can be correctly associated with the records that possess modified information (i.e., the ones extracted in the "Extract I, U, D" transition). This is accomplished through the arc expression (#tid tlr, #tname tlr). At the same time, the progress of the CDC process is once again recorded. Note that the TLog Prog is not updated for the log records extracted in the "Extract I, U, D" transition as they aren't fully processed until they are used to update the corresponding audit table. In this module, the Extract BEGINS transition has the lowest priority (P_3), which makes that a new transaction is only processed after the records of the current transaction have been fully processed in the remaining modules.

5.2 The Decode Module

After LOP_BEGIN_XACT and LOP_COMMIT _XACT log records have been fully processed, the next step is to process the remaining records extracted in the previous module, i.e., the ones related with LOP_INSERT_ROWS, LOP_ MODIFY_ROWS or LOP_DELETE_ROWS operations. These are the records that posses hexadecimal values in the Row Log Contents attributes, rather that 'NULL' values, and this is where the information about the inserted, deleted or updated data resides. The actual transformation of the hexadecimal values into the modified rows is out of the scope of this study, which is focused on the interpretation of the transaction log for CDC and the update of the audit tables once the modified data has been decoded. Nevertheless, in the physical design of this ETL process, these transformations would be represented by a decoding function that is simulated in this module.

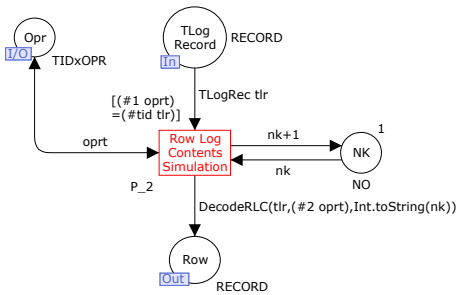

Figure 3: The Decode module.

The Decode module (Figure 3) is used to simulate the transformation of hexadecimal values into the modified rows and is formed by a single transition and four places. The place *TLog Record* acts as an input port and is used to model the log records with modified data extracted in the previous module, while the place *Row* is used to model a log record after the *Row Log Contents* attribute has been decoded. This new record, referred to as decoded record, contains additional attributes that represent the modified row in the operational source:

```
colset DECREC = record lsn:NO * tid:NO *
src:ST * opr:ST * tn:ST * nk:ST *
atb:ST;
```

This new type of record maintains the *lsn*, *tid* and *opr* fields from the original log record. The *nk* and *atb* fields are strings used to represent the natural key and a second attribute resulting from the Row Log Contents decoding. The fields *src* and *tn*, both strings, represent the name of the operational source and the name of the audited table respectively; these values are derived from the *aun* field of the original log record. The place *Opr*, an input/output port, holds tokens with information on the names of the transactions, plus their IDs, extracted from the LOP_BEGIN_XACT log records in the first module.

The guard expression `[(#1 opr) = (#tid tlr)]` is used so that the correct transaction name ('Insert', 'Delete' or 'Update') is passed to the transition and used later in this module. To simulate the decoding of the Row Log Contents the place NK, with colour set NO and initial marking '1', is used together with the function *DecodeRLC* in the arc expression that leads to *Row*. In each marking the variable *nk* is incremented and passed to the transition trough the arc expression *nk*+1; the incremented integer is then converted to a string and used as parameter in the *DecodeRLC* function. This function takes three parameters: the log record represented by the *tlr* variable, the name of the transaction, and the incremented value of *nk*.

```
fun DecodeRLC(tlr:TLOGREC,tranName,nk) =
    1`DecRec{lsn =(#lsn tlr), tid =
    (#tid tlr),
           src = "srcDB", opr = tranName,
           tn = substring((#aun tlr),4,2),
           nk = "nk"^nk, atb = "atb"^nk};
```

The new decoded record maintains the values in the *lsn* and *tid* fields of the log record. The value of the *opr* field is substituted by the transaction name passed in the *tranName* variable so that the operation is displayed as 'Insert', 'Delete' or 'Update'. The name of the audited table in the operational source is extracted from the *alloc* unit name field (*aun*) of the original log record, which contains the schema name and the name of the table where the operation occurred. This is done through the substring function and the resulting string is saved into the *tn* field of the decoded record. The name of the source database can be obtained either by querying the operational source for the current database name, or by executing the stored procedure *sp_Msforeachdb*, comparing the name of the audited table with the column name in either *sys.tables* or *sysobjects* tables. This check all the names of the tables in the existing databases and return the name of the database that has the requested table name. This procedure can be implemented as:

```
exec master.dbo.sp_msforeachdb
    "USE [?]
    SELECT db_name()
           FROM sysobjects
           WHERE name='T1'"
```

In the model we created, the execution of such a query is not possible and the *DecodeRLC* function is also used to simulate this operation by updating the *src* field. To simulate the decoding of the natural key value of the source record, the *nk* variable passed as a function parameter is concatenated with the string "nk" so that a different natural key value (e.g. 'nk1', 'nk2', 'nk3') is created for each record. The same happens with the *atb* field.

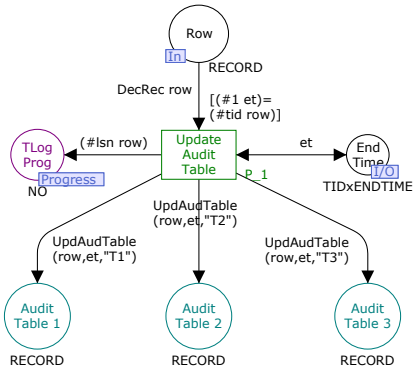

Figure 4: The Audit module.

5.3 The Audit Module

The Audit module (Figure 4) is our last module. It assigns the timestamps of the transactions to the correct decoded records, as well as their insertion in the corresponding audit tables. The place Row is an input port and has tokens representing the decoded records, the place *End Time* is an input/output port and has tokens with information on *ID* of each transactions and the corresponding timestamp, and the fusion place *TLog Prog* is once again used to record the log progress. The places Audit Table 1-3 are used to model the audit tables of three different relations in the operational sources and receive tokens representing the final audit records. The colour set *AUDREC* is used to model this type of records and it is defined as:

```
colset AUDREC = record src:ST * dtm:ST *
    opr:ST * nk:ST * atb:ST;
```

This record maintains the *src*, *opr*, *nk* and *atb* fields and has an extra field, *dtm*, which represents the timestamp of the operation in the operational source. The single transition *Update Audit Table* receives a decoded record through the arc expression *DecRec* row and the corresponding timestamp through the variable *et* in the arc that connects the place End Time with the transition. For the transition to be enabled, the guard expression `[(#1 et) = (#tid row)]` must be true, meaning that there is a transaction *ID* in the place *End Time* that matches the transaction *ID* of one of the decoded records in Row. The *UpdAudTable* function creates a fresh audit record with the corresponding timestamp and inserts it in the correct audit table:

```
fun UpdAudTable(r:DECREC,et:TIDxENDTIME,tname)=
   if (#tn r )= tname
      then CreateAudRec(r,et)
      else empty
```

This function receives the decoded record *r*, the pair transaction *ID* – *end time*, represented by the

variable *et* and the name of the table, represented by the variable *tname*. Then, it compares the value in the *tn* field of the decoded record with the value in the *tname* variable passed as a parameter; if they match, then the correct audit table is being updated and a new audit record can be created through the function *CreateAudRec*:

```
fun
CreateAudRec(r:SRCREC,et:TIDxENDTIME)
     = 1`AudRec{src = (# src),
           dtm = (#2(et)),
           opr = (#opr r), nk = (#nk r),
           atb = (#nk r)};
```

The *CreateAudRec* function creates a fresh audit record. The only piece of information missing is the timestamp of the operation, passed as a parameter through the variable et. By using these two functions and a single transition, it is possible to insert the final audit record in the correct audit table. The final step of this module's process is to update the fusion place *TLog Prog* with the *LSN* of the decoded records used to generate the audit records.

6 CONCLUSIONS

In this paper, a CDC mechanism is presented as a case study to demonstrate the viability of adopting the CPN modelling language to model and validate ETL processes and tasks. The CPN modelling language allows a DWS to be hierarchically modelled, i.e., one can build a module that can be composed of several smaller modules representing each populating process. This is very useful when modelling these types of systems as they can be greatly simplified by adding different abstraction levels, making it easier for the designer to build large models while improving their readability and understanding. At the DWS conceptual design stage, a correct ETL system specification, written in the CPN modelling language, allows for reducing the occurrence of design errors and their impact in the development of the entire DWS, once through simulation we can detect them previously. The CDC process described here can be used as an independent module and used together with other modules that represent different tasks to build higher-level models of ETL systems. The described model is based in a general CDC mechanism of a DBMS and some predefined values and environment configuration. This is not enough for supporting real world ETL systems modelling, but it helps a lot. However, we need to extend it in the short term so that it can be applied to a wide variety of ETL scenarios and, consequently, to be used as an effective modelling tool for DWS.

REFERENCES

Abelló, A., Samos, J., and Saltor, F. YAM2: a multidimensional conceptual model extending UML. Information System, 31(6), 541–567, 2006.

Cherkasova, L., Kotov, V., and Rokicki, T., On Scalable Net Modelling of OLTP. *In Proceedings of the 5th International Workshop on Petri Nets and Performance Models*. IEEE Computer Society Press, 270–279, 1993.

English, L. P., Improving data warehouse and business information quality: methods for reducing costs and increasing profits. *John Wiley & Sons, New York*, NY, USA, 1999.

Figueiredo, J. C. A. D. and Kristensen, L. M., Using Coloured Petri Nets to Investigate Behavioural and Performance Issues of TCP protocols. In Department of Computer Science, *Aarhus University*. 21–40, 1999.

Golfarelli, M., The DFM: A Conceptual Model for Data Warehouse. *Encyclopedia of Data Warehousing and Mining (2nd Edition),* John Wang (ed.), IGI Global, 2008.

Golfarelli, M., Rizzi, S., Data Warehouse Design: Modern Principles and Methodologies, *1st ed. McGraw-Hill, New York*, NY, USA, 2009.

Inmon, W., Building the Data Warehouse, John Wiley & Sons, 1996.

Jensen, K., An introduction to the practical use of coloured petri nets. In Lectures on Petri Nets II: *Applications, Advances in Petri Nets*, Springer, London, UK, 237–292, 1998.

Jensen, K., Krinstensen, L., Coloured Petri Nets: Modeling and Validation of Concurrent Systems. Springer, New York, NY, USA, 2009.

Kimball, R., Caserta, J., The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. John Wiley & Sons, 2004.

Muñoz, L., Mazón, J., Trujillo, J., Automatic generation of ETL processes from conceptual models. In Proceeding of the ACM *12th International Workshop on Data warehousing and OLAP DOLAP 09*, ACM Press, 2009.

Petri, C.A., Kommunikation mit Automaten. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962. English translation: *Technical Report RADC-TR-65-377*, Griffiths Air Force Base, New York, Vol. 1, Suppl. 1, 1966.

Vassiliadis, P., Simitisis, A., Skiadopoulos, S.. Conceptual modeling for ETL processes. *In Proceedings of the 5th ACM International Workshop on Data Warehousing and OLAP (DOLAP '02)*. ACM Press, 14–21, 2002.